コア・イメージを利用したアルゴリズム教育の試み

Trial for Algorithm Education using Core Images of Algorithms

川口　順功
Junkoh KAWAGUCHI

（平成24年10月 9 日受理）

アルゴリズムを教育で扱うとき、それを使って問題を解く（プログラム作成も含む）ことが優先されがちであるが、アルゴリズムを学ぶ意義は、その本質的な理解と応用にもある。アルゴリズムの本質的な理解と応用の鍵は、その構造の理解にあると考えられる。したがって、アルゴリズムを教育で扱うとき、アルゴリズム構造を理解させることが重要なテーマとなる。

本稿では、アルゴリズム構造の理解をアルゴリズムのコア・イメージの理解と位置づけ、コア・イメージによるアルゴリズムの本質的な理解と応用について考える。これまで試みてきたコア・イメージを利用したアルゴリズム教育の実践例を紹介しながら、大学でのアルゴリズム教育について考察する。

1．まえがき

「A地点からB地点に車で行くのに、一般道路を平均時速40km、高速道路を平均時速80kmで移動したところ、 4 時間で到着し、走行距離は220kmであった。一般道路と高速道路をそれぞれ何時間走ったか？」という問題は、鶴亀算のアルゴリズムで解けることがすぐにわかるだろうか。もっと端的に言えば、この問題が鶴亀算であることを直観的にわかるかである。鶴亀算と直観的にわかる、あるいは、鶴亀算と言われたとき直観的にそれとわかるにはどうすればよいか。これが、本研究の出発点である。この問題がアルゴリズム構造に基づいて作成されたものであることを考えると、直観的な理解にはアルゴリズム構造の理解が必要と考えられる。

アルゴリズムを教育で扱うとき、それを使って問題を解く（プログラム作成を含む）ことが優先されがちである。しかし、アルゴリズムを学ぶ意義は、問題解決の手順としてのアルゴリズムの理解だけではなく、そのベースとなるものの捉え方や考え方などを含めたその本質的な理解と他の問題への応用にもある。アルゴリズムの本質的な理解と応用の鍵は、その構造の理解にあると考えられる。したがって、アルゴリズムを教育で扱うとき、アルゴリズム構造を理解させることが重要なテーマとなる。

本研究の目的は、大学におけるアルゴリズム教育において、問題を解くとき必要となるアルゴリズムをその構造から理解させ、その本質的な理解と応用につなげることである。本研究では、アルゴリズム構造の理解をアルゴリズムのコア・イメージ（詳細は後述）の理解と位置づける。コア・イメージは、アルゴリズムをイメージ化したもので、その構造が反映されたものでなければならない。そして、アルゴリズム名とともに想起され、アル

ゴリズムの本質的な理解と応用につながるものとする。つまり、アルゴリズム構造の理解の根拠をコア・イメージの理解に置き、その理解をアルゴリズムの本質的な理解と応用につなげるという試みである。

本稿では、これまで試みてきたコア・イメージを利用したアルゴリズム教育の実践例[1][2][3][4]を紹介しながら、大学でのアルゴリズム教育について考察する。

2．アルゴリズムのイメージ化とコア・イメージについて

アルゴリズムをイメージ化するとはどういうことか。「イメージ（image）」は、一般的には、「心の中にうかべる姿・像、感覚像、心象」[5]などの意味で使われる。本稿では、対象とするアルゴリズム名を聞いたとき、想起される図、式、グラフ、文章、または、それらを組み合わせたものを、そのアルゴリズムのイメージとし、そのようにイメージされるものを作成することをイメージ化と呼ぶ。ただし、イメージ化されたものは、アルゴリズム構造が反映されていなければならない。したがって、本稿では、アルゴリズムのイメージ化は、一般的に言葉で定義または説明されるアルゴリズムを図や式などで表現し、それによってアルゴリズム構造を想起できるようにすることを意味する。表現されたものをイメージ図と呼び、特に、対象となるアルゴリズム名を聞いたとき、すぐに想起されるメインとなるイメージ図を「コア・イメージ（core image）」と呼ぶ。したがって、アルゴリズムをイメージ化するということは、アルゴリズムのコア・イメージを作成することだと考えてよい。

筆者が「アルゴリズムのコア・イメージ」に注目し始めたのは、言葉に対する（一般的な意味での）イメージの重要性を強く意識するようになってからである。例えば、「オンライン（online）」と「オンデマンド（on demand）」にある「オン（on）」のイメージである。「オン」の本質的な意味はモノとモノの「接触」という位置関係[6]であり、「オン」のイメージも「接触している状態」でなければならない。このことを理解していると、「オンライン」は、回線との非接触（off）を意味する「オフライン（offline）」との対比で意味が明確になり、「オンデマンド」の「オン」は、「接触 → モノ（コト）とモノ（コト）の接触でつながる → ～と同時に」とイメージを展開すれば、「要求と同時に」という意味になることがわかる。したがって、「オン」のイメージによる本質的な理解が、「オンライン」と「オンデマンド」という2つの言葉の理解と大きく関わってくる。

また、「率」や「関数」などの言葉に対しても、明確なイメージを持っている学生は意外に少ない。ある授業で、学生に「円周率」という言葉を知っているか質問したことがある。ほとんどの学生が知っているというので聞いてみると、「3.14」と答えるだけで、「円周／直径（円周の直径に対する割合）」と答える学生は少ないことがわかった。このような事態が起こる原因は、実は円周率だけの問題ではなく、「率」という言葉のイメージをしっかり持っていないからである。「率」という言葉を聞いたら、「率＝A／B（AのBに対する割合）」をイメージし、さらに、このイメージと同時に「率を上げるためには、分子のAを増やすか、分母のBを減らすかである。ただし、Bには下限がある場合がある」ことを、率に対するイメージとして持つように指導する必要がある。また、「関数」という言葉も、「y＝f（x）」という式とグラフだけではなく、例えば、図2-1のような図[7]

（原著ではｙ＝ｆ（ｘ）のイメージに合わせｘとｙの位置と矢印の向きが逆である）をイメージとして持ち、「ｘの値が与えられると、それに対してｙの値がただ一つ決まる」ということを理解しておく必要がある。

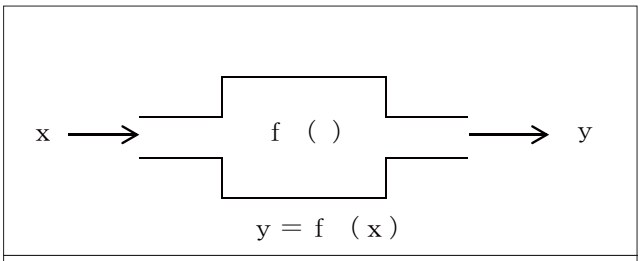

図2-1 関数のイメージ図

本研究でも、アルゴリズム名を聞いたとき想起される言葉としてのイメージを意識し、それをアルゴリズムのコア・イメージと考える。上述した一般的な言葉のイメージとの違いは、コア・イメージがアルゴリズム構造を反映し、その理解の根拠とされることである。

筆者は、アルゴリズム教育で扱うアルゴリズムについては、できるだけコア・イメージを作成し、それを使って授業を実施している。コア・イメージを利用したアルゴリズム教育には、アルゴリズム構造の理解だけでなく、いくつかの効果が期待される。G.ポリア著「いかにして問題をとくか」[8]の中には、その根拠になると思われるいくつかのヒントがある。その中で、著者は、問題を解くステップを「問題を理解すること」「計画をたてること」「計画を実行すること」「ふり返ってみること」の４つに分け、さらに細かく15項目にわたって問題の考え方や解き方のヒントについて書いている。15項目の中に「（問題を理解すること）未知のものは何か。与えられているもの（データ）は何か。条件は何か」「（問題を理解すること）図を書け。適当な記号を導入せよ」「（計画をたてること）前にそれをみたことがないか。又は同じ問題を少しちがった形でみたことがあるか」「（計画をたてること）似た問題を知っているか。役に立つ定理を知っているか」「（ふり返ってみること）他の問題にその結果や方法を応用することができるか」などがある。これらもヒントにして、コア・イメージを利用したアルゴリズム教育には、アルゴリズム構造の理解をベースに、「図や記号を使った問題の理解」「類似問題の想起」「結果や方法の他の問題への応用」「記憶の定着」などの効果が期待される。

３．コア・イメージを利用したアルゴリズム教育の実践例とその考察

本稿で対象とするアルゴリズムは、プログラミングを前提としたアルゴリズム（以下、プログラミング的アルゴリズム）と「鶴亀算」や「濃度算」などのような数学（算数）の問題を解くときのアルゴリズム（以下、数学（算数）的アルゴリズム）である。

筆者は、アルゴリズム教育として「情報構造基礎」「アルゴリズム応用論」などの科目を担当している。大学で行うアルゴリズム教育は、基本的にプログラミング教育の中に位置づけられ、プログラミングを前提としたプログラミング的アルゴリズムを対象とする。しかし、筆者は、それだけではなく、アルゴリズム構造の理解という観点から数学（算数）

的アルゴリズムも必要と考え、それらを授業の中に取り入れている。

ここでは、アルゴリズム構造を反映させて作成したコア・イメージを取り上げ、そのアルゴリズム教育での利用について考察する。プログラミング的アルゴリズムでは、「２分探索」「クイックソート」「マージソート」「再帰的アルゴリズム」の実践例を取り上げる。これらのアルゴリズムはすでに確定しているものであるので、「アルゴリズムの基本的な考え方」「具体的なアルゴリズム」「コア・イメージの説明」「プログラム」の順に説明する。ただし、「再帰的アルゴリズム」は他のアルゴリズムのベースとなるものであるので、「アルゴリズムの基本的な考え方」と「コア・イメージの説明」のみとする。

数学（算数）的アルゴリズムでは、「鶴亀算」「流水算」「濃度算」「ｎ進法」の実践例を取り上げる。これらのアルゴリズムは具体的なアルゴリズムがコア・イメージと関連するので、「問題の背景とアルゴリズムの基本的な考え方」「コア・イメージの説明」「具体的なアルゴリズム」「アルゴリズムに基づく解」の順に説明する。

３－１．プログラミング的アルゴリズム

プログラミング的アルゴリズムは、言葉で定義または説明されるものが多い。アルゴリズム辞典[9]では、アルゴリズムは言葉による定義または説明が中心で、一部補足的に説明図が用いられているものもある。また、アルゴリズム関連の図書[10][11][12]では、プログラムを前提とした図(配列など)を用いた説明が中心で、言葉で明確に定義しているアルゴリズムも意外に少ない。これらの図書の目的からアルゴリズムの理解をプログラム表現と位置づけていると思われるので、このような形になるのはやむを得ないことである。

ここでのコア・イメージは、アルゴリズムの説明に使われているものを元に作成されるものが多い。しかし、説明に使われているものがプログラム表現に重点を置いていたり、複数の図にまたがっているような場合は、アルゴリズム構造を反映させる工夫が必要となる。コア・イメージは、アルゴリズムの内容により、「プログラム構造」または「処理されるデータの流れ」のいずれかに焦点を当てて作成するが、両方に捉えられるものもある。プログラム構造に焦点を当てる場合、プログラム表現も考慮される。

３－１－１．２分探索

２分探索（binary search）は、整列（ソート）されたデータから目的のデータを探索するアルゴリズムである。アルゴリズム辞典[9]には、「データが大きさの順に並んだ表の探索法の一つ。バイナリーサーチともいう。探索(サーチ)するデータを、順に並んだデータの中央に位置するデータと比較すると次のいずれかになる。(1)探索データが小さい。(2)一致する。 (3)探索データが大きい。(2)の場合はデータを見つけたことを意味するので探索はここで終わる。(1)の場合は、中央のデータより後ろには、大きなデータしかないので、求めるデータは前半にあることがわかる。同様に(3)の場合は後半にあることがわかる。いずれにしても、探索対象として残るデータの個数は最初の半分になる。データが見つかるか、探索の対象となるデータがなくなるまで、この操作を繰り返す」と書かれている。

筆者は、コア・イメージを使う以前は、図3-1に示すような具体的な配列を使ってアルゴリズムを説明していた。図の■は「未探索のデータ」、×は「探索対象外のデータ」を

表す。この図は、図3-1(a)の配列から13（キー値）を探索する場合の例で、具体的に説明すると次のようになる。

- 中央の値 9 と比較する（図3-1(b)）。
- 「(中央の値) 9 ＜ 13（キー値)」より、探索範囲は図3-1(c)の■部分となる。
- 図3-1(d)の中央の値13と比較し、一致するので探索成功で終了となる。

ここで考えたいことは、このような説明図がアルゴリズム構造の理解とうまくリンクできるかどうかである。少なくとも、次に示すコア・イメージの方が、アルゴリズム構造の理解とうまくリンクできるのではないかと期待している。

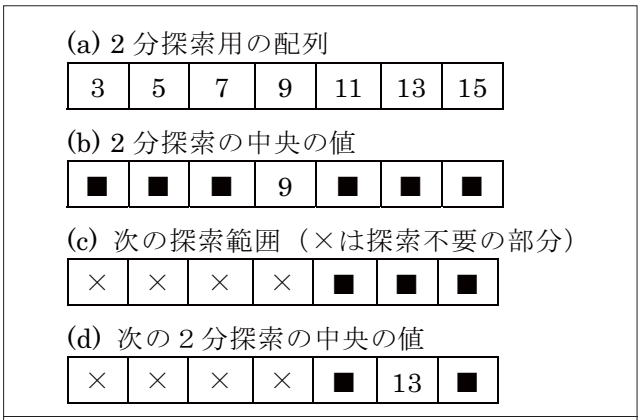

図3-1　2 分探索の例

（アルゴリズムの基本的な考え方）

昇順（降順の場合もある）にソートされた探索対象領域の中央の値と探索keyの値を比較し、一致したら探索成功で探索終了、(探索key ＜ 中央の値）のとき中央より前の部分が次の探索対象領域に、(中央の値 ＜ 探索key）のとき中央より後の部分が次の探索対象領域になる。探索対象領域がなくなるまで以上の処理を繰り返す。

（具体的なアルゴリズム）

以下の処理を探索成功か、探索対象領域がなくなる（探索失敗）まで繰り返す。

① 探索対象領域の中央の値と探索keyを比較する。
②（探索key ＝ 中央の値） → 探索終了。
③（探索key ＜ 中央の値） → 探索対象領域を中央の値より前の部分にして①へ。
④（探索key ＞ 中央の値） → 探索対象領域を中央の値より後の部分にして①へ。
（ただし、③④で探索対象領域がない場合は、探索失敗で終了）

（コア・イメージの説明）

図3-2は、2 分探索のコア・イメージである。これは、プログラム構造に焦点を当てて作成されるものであるが、一部のデータの流れに焦点を当てているとも考えられる。基本的に、探索keyと配列の中央の値Cを比較し、(key ＝ C）のとき探索成功で終了、(key ＜

C）のとき探索範囲が前半分、（C ＜ key）のとき探索範囲が後半分になることが理解できればよいので、３つの処理に分岐するプロセスが、次の探索範囲も含めて理解できる構造であればよい。このコア・イメージには、このことが反映されている。

ここでアルゴリズム構造の理解として、見落とされ勝ちになるのが、配列の扱われ方である。線形探索では１要素ずつ線形的に扱われるのに対し、２分探索では集合的に扱われる[1]ことである。このように配列が集合的に扱われることが、２分探索の方が線形探索より探索処理が速い要因であり、このことがコア・イメージには反映されている必要がある。このコア・イメージでは、配列の推移図からこのことがわかる。そして、アルゴリズムの評価に使う時間計算量O（order）は、線形探索がO(n)（データ件数nに比例）であるのに対し、２分探索がO(log n)（log nに比例）であることが、このコア・イメージで説明できる。

また、処理される全体のデータの流れに焦点を当てると、図3-3に示す２分探索木のイメージ図になる。２分探索木のノード（節）になるのが配列の中央の値で、各ノードの左部分木は図3-2の左側の部分配列、右部分木は同じく右側の部分配列となる。それぞれの部分木のルート（根）になるのが部分配列の中央の値になることが理解できれば、２分探索は、２分探索木による探索と本質的に同じであることが理解できる。

（プログラム）

図3-4は、上述したアルゴリズムに基づいて作成されたプログラム[10]である。「pl（探索範囲の左端）」「pr（探索範囲の右端）」「pc（探索範囲の中央値）」として、探索keyと中央の値C(=a[pc])の関係によって分岐するプロセスが、コア・イメージのとおりにプログラムで表現されていることがわかる。なお、このプログラムは、do-while文を使っているが、データ件数が0の場合があり得る場合は、while文を使わなければならない。

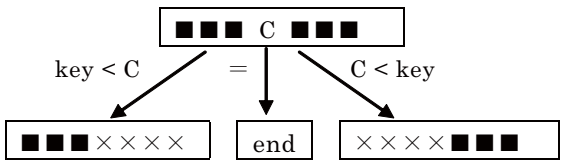

図3-2　２分探索のコア・イメージ

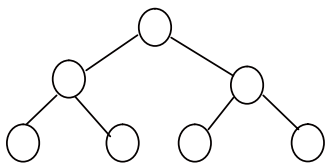

図3-3　２分探索の展開イメージ図

３－１－２．クイックソート

クイックソート（quick sort）は、ソートアルゴリズムの一つで、「ピボット（pivot：基軸となる値）により、データをピボットより小さい値のグループと大きい値のグループの２つに分け、さらに分けた２つのグループに同じ操作を再帰的に適用する」という考え方に基づく。アルゴリズム辞典[9]には、「配列のある値より大きい値の部分と小さい値の部分の二つに分け、それぞれを別々に整列（ソート）するという方針をとる。二つの部分の整列にも同じアルゴリズムを使うので、再帰的な処理になる。整列の問題に分割統治法を適用した例になっている」と書かれている。

```
int bin_search(const int a[], int n, int key)
{
      int   pl = 0;
      int   pr = n - 1;
      int   pc;
      do {
            pc = (pl + pr) / 2;
            if (a[pc] == key)
                  return (pc);
            else if (key < a[pc])
                  pr = pc - 1;
            else
                  pl = pc + 1;
      } while (pl <= pr);
      return (-1);
}
```

図3-4　2分探索のプログラム例

（アルゴリズムの基本的な考え方）

再帰呼び出しのプログラムを前提にしたアルゴリズムで、これを理解するには再帰的アルゴリズムの理解が必要となる。基本的には、ソート対象領域をピボットによりピボット以下とピボット以上の領域に分け、その分けた2つの領域に同じ操作を再帰的に適用するという考え方である。ピボットの値は、領域分割アルゴリズム（この理解も必要）によりどちらかの領域に入る。

（具体的なアルゴリズム）

再帰呼び出しを使う関数としての具体的なアルゴリズムは、以下のようになる。

①配列のデータからピボットを選ぶ。
（配列の中央の値を取る場合が多い）
②領域分割アルゴリズムにより、ピボット以下のデータとピボット以上のデータの2つのグループに分ける。（ピボットのデータは状況によりどちらかのグループになる）
③ピボット以下(前方)のグループのデータ数が2以上なら、①〜④を再帰的に繰り返す。
④ピボット以上(後方)のグループのデータ数が2以上なら、①〜④を再帰的に繰り返す。
（③④で、再帰的アルゴリズムを使う）

（コア・イメージの説明）

図3-5は、処理される配列のデータの流れに焦点を当てたコア・イメージである。一方で、プログラム構造に焦点を当てたコア・イメージも考えられる。これについては後述するが、プログラム表現に重点を置けば、これをコア・イメージとすることもできる。しかし、グループ分けに使われる領域分割アルゴリズム、データ展開の順序、高速アルゴリズムである根拠などを意識して、図3-5をクイックソートのコア・イメージとしている。

処理されるデータに焦点を当てるので、基本的にピボットにより2つに分割されていく配列の状態がわかればよい。つまり、2分木のイメージである。しかし、このコア・イメージでは、配列の要素が1個になったものはそれ以上分割する必要がなく最終状態であるこ

とを示すために、2分木から切り離してある。これは、配列の状態によっては2分木ではなく、配列の要素が1個になったものが中央に残り、3つ以上の分岐になる場合があるからでもある。これはピボットによる領域分割アルゴリズムに基づくのであるが、このアルゴリズムの説明は省略する。

2分木の展開の順序の理解も必要と考えられる。この展開は、再帰的アルゴリズム（後述）により深さ優先順探索（木のルートから左方向に迷路をたどるように探索する）の順で行われる。コア・イメージには展開の順序は示していないが、これを使って具体的な展開の順序を説明することができる。

配列の値を明示しているのは、全体的な展開とともに、ピボット（アンダーラインの数値）による配列の領域分割アルゴリズムの理解も視野に入れているからである。学生には、このコア・イメージを自分で作成できるようになることが求められる。アルゴリズム構造の理解がコア・イメージの理解とリンクする典型的な例と考えてよい。

クイックソートは、現在、最高速のアルゴリズムと言われている。その主な要因は、配列を集合的に扱うことと、領域分割アルゴリズムによりデータ交換するときのデータ移動を大きくできることである。このことも理解させる必要があるが、コア・イメージには、このことも考慮されている。クイックソートの時間計算量は$O(n \log n)$で、単純選択ソート、単純交換（バブル）ソート、単純挿入ソートなどの時間計算量が$O(n^2)$であることは、この配列の扱い方の違いによるが、このことはコア・イメージで説明できる。

このコア・イメージと図3-10（後述）に示すクイックソート用プログラム展開イメージ図（再帰的アルゴリズムのコア・イメージに基づいて作成）をリンクさせると、再帰的アルゴリズムに基づく再帰呼び出しが具体的に追跡できるので、このコア・イメージは再帰的アルゴリズムの具体的な理解にも使える。

（プログラム）

図3-6は、上述したアルゴリズムに基づいて作成されたプログラム[10]である。プログラム中の①～④の番号は、前述したアルゴリズムの番号と対応している。再帰呼び出しは、③と④の部分である。このプログラムは、処理される配列のデータの流れに焦点を当てたコア・イメージではなく、図3-10（後述）で示すクイックソート用プログラム展開イメージ図を参照する方が理解しやすい。

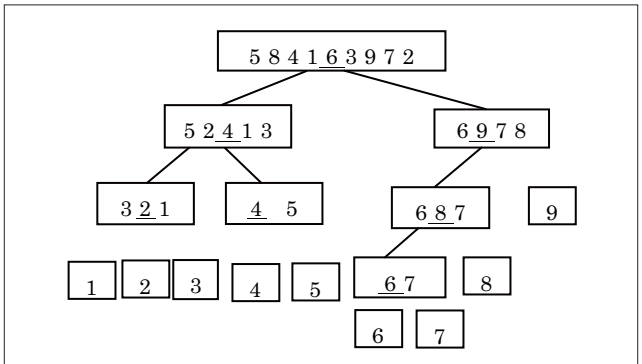

図3-5　クイックソートのコア・イメージ

```
void quick(int a[], int left, int right)
{
      int pl = left, pr = right, temp;
①    int x = a[(pl+pr)/2];
②    do {
            while (a[pl] < x) pl++;
            while (a[pr] > x) pr--;
            if (pl <= pr) {
                  temp = a[pl];
                  a[pl] = a[pr];
                  a[pr] = temp;
                  pl++;    pr--;
            }
      } while (pl <= pr);
③    if (left < pr)    quick(a, left, pr);
④    if (pl < right) quick(a, pl, right);
}
```

図3-6 クイックソートのプログラム例

3 - 1 - 3．マージソート

マージソート（merge sort）は、「配列を前半部と後半部の 2 つに分け、それぞれをソートした後マージする。分けた各部には、再帰的にマージソートの考え方を適用する」という考え方に基づく。アルゴリズム辞典[9]には、「入力データを二つに分割し、それぞれ別々にソート(整列)し、その結果を併合（マージ）することによって、もとの入力データをソートする方法。二つに分割されたおのおのの組はさらに分割することによって、マージソートアルゴリズムを再帰的に適用する。分割を繰り返した結果、要素を一つのみもつ列の集まりになる。この状態から複数の列（要素が一つまたはソートされている）を併合して一つの列にする操作を再帰的に適用することで、最終的に要素すべてをソートした列一つになる」と書かれている。

（アルゴリズムの基本的な考え方）

クイックソートと同様に、再帰呼び出しのプログラムを前提にしたアルゴリズムで、これを理解するには再帰的アルゴリズムの理解が必要となる。ソート対象領域を単純に 2 分し、2 つの領域に対して同じマージソートのアルゴリズムを再帰的に繰り返してソートし、その 2 つの領域を併合して一つのソート済領域にするという考え方に基づく。

（具体的なアルゴリズム）

再帰呼び出しを使う関数としての具体的なアルゴリズムは、以下のようになる。

①配列Ａが 2 つ以上のデータの場合、以下の処理を行う。

（1 つのデータの場合、何もしない）

②配列Ａを 2 つの配列Ａ1、Ａ2に分割する。（基本的には中央で分ける）

③配列Ａ1に対して①～⑤を再帰的に繰り返す。

④配列Ａ2に対して①～⑤を再帰的に繰り返す。

⑤配列Ａ1と配列Ａ2をマージして、その結果を配列Ａに戻す。

（③④で、再帰的アルゴリズムを使う）

（コア・イメージの説明）

図3-7は、処理される配列のデータの流れに焦点を当てたコア・イメージである。これはアルゴリズムの説明でよく利用されるもので、アルゴリズム辞典[9]でも説明図として掲載されている。これもクイックソートと同様に、プログラム表現に重点を置いたコア・イメージが考えられるが、ほぼ同じような理由で、これをコア・イメージとしている。

クイックソートと同様に、基本的には、2つに分割され、1つに併合されていく配列の状態がわかればよい。つまり、2分木（併合は展開の向きが逆となる）のイメージである。さらに、コア・イメージでわかるように、分割と併合は、分割終了の所を中心に上下で対象になる。分割と併合の順序の理解も必要である。2分木の展開は、単純に2分するだけであり、クイックソートと異なり常に2分木で配列の要素が1個になるまで行われる。展開の順序は、クイックソートと同様に再帰的アルゴリズムにより深さ優先順探索の順であり、併合は同じレベルの2つの併合が終了した時点で次の併合になる。ここでも配列の値を明示しているが、その目的はクイックソートの場合と同じである。

マージソートも、時間計算量がO(n log n)の高速のアルゴリズムである。その要因の一つは、クイックソートと同様に配列を集合的に扱うことであり、これもこのコア・イメージで説明できる。

クイックソートと同様に、マージソート用プログラム展開イメージ図を作成し、それとコア・イメージをリンクさせると、再帰的アルゴリズムに基づく再帰呼び出しが具体的に追跡できるので、このコア・イメージも再帰的アルゴリズムの具体的な理解に使える。特に、分割と併合のタイミングの理解には有効と思われる。

（プログラム）

図3-8は、上述したアルゴリズムに基づいて作成されたプログラム[10]である。プログラム中の①～⑤の番号は、前述したアルゴリズムの番号と対応している。再帰呼び出しは、③と④の部分である。このプログラムも、クイックソートの場合と同様に、マージソート用プログラム展開イメージ図を作成し、それを参照する方が理解しやすい。

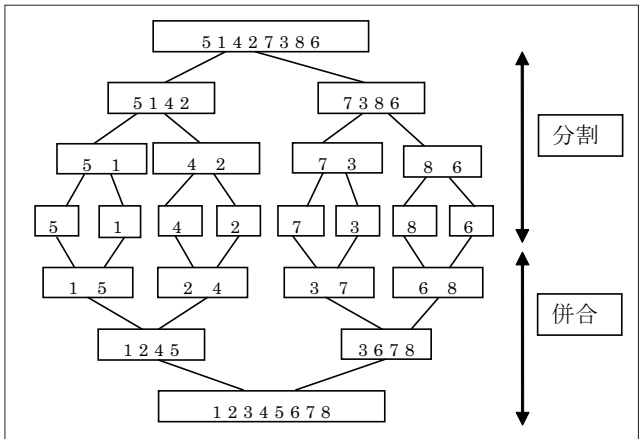

図3-7　マージソートのコア・イメージ

```
static int            *buff;
static void __mergesort(int a[], int left, int right)
{
①   if (left < right) {
②         int       center = (left + right) / 2;
          int       p = 0, i, j = 0,    k = left;
③         __mergesort(a, left, center);
④         __mergesort(a, center + 1, right);

⑤         for (i = left; i <= center; i++)
                buff[p++] = a[i];

          while (i <= right    &&    j < p){
                a[k++] = (buff[j] <= a[i]) ? buff[j++] : a[i++];
          }
          while ( j < p)
                a[k++] = buff[j++];

    }
}
int mergesort(int a[], int n)
{
    if ((buff = (int *)calloc(n, sizeof(int))) == NULL)
          return (-1);
    __mergesort(a, 0, n - 1);
    free(buff);
    return (0);
}
```

図3-8　マージソートのプログラム例

3 - 1 - 4．再帰的アルゴリズム

再帰的アルゴリズムは、「ある問題を部分問題に分割したとき、その部分問題が元の問題と同じ構造であることを利用して、部分問題を元の問題と同じアルゴリズムを使って解くアルゴリズム」である。プログラムでは、自分の関数から自分自身を呼び出す再帰呼び出しの形になる。アルゴリズム辞典やアルゴリズムの関連図書には、再帰的アルゴリズムとしての説明ではなく、再帰呼び出しを説明する形で記述されている。

（アルゴリズムの基本的な考え方）

クイックソートやマージソートの例のように、同じような処理を再帰的に繰り返すとき使うアルゴリズムで、特に再帰的に繰り返す処理のネストの最大値がわからないとき有効である。このアルゴリズムは、基本的にはスタック処理や繰り返し文などを使って実現できるが、そのプログラムは一般的には複雑になる。逆に、このアルゴリズムをうまく利用するとアルゴリズム自体が簡潔になり、プログラム表現も簡潔になる。

（コア・イメージの説明）

図3-9は、再帰的アルゴリズムのコア・イメージである。この図では、全体の問題がA、B、Cの 3 つの部分問題からなり、問題C（C1、C2）が元の問題と同じ構造になる部分問題であることを表している。この条件のもとで、アルゴリズム構造としての再帰呼び出しがわかればよい。再帰呼び出しが一つ深まるごとに、部分問題のレベル番号を加算するようにしてある。これは、構造的には同じであるが、問題の内容（処理対象とするデータな

ど）が異なることを示すためである。

矢印で示される再帰呼び出しは、プログラムでは自分の関数から自分自身を呼び出す形であるが、実体は自分と同じ関数（コピー関数）を呼び出すことであるので、このことが捉えられるように作成されている。要するに、基本的に上から下に逐次実行される処理が、矢印で示された所で自分のコピー関数を呼び出し、その関数の処理が終了した時点で呼び出し元に戻るという、一般的な関数呼び出しの形であることが理解できればよいので、そのことが意図されている。

図3-10は、クイックソートを再帰的アルゴリズムで展開したクイックソート用プログラム展開イメージ図である。この図は、クイックソートのアルゴリズムと再帰的アルゴリズムのコア・イメージから作成できる。①～④は、前述したアルゴリズムの番号と対応している。前述したように、この図3-10と図3-5のクイックソートのコア・イメージを併用すれば、再帰的アルゴリズムによる処理の展開を具体的に追跡でき、再帰的アルゴリズムの理解も高まると考えられる。

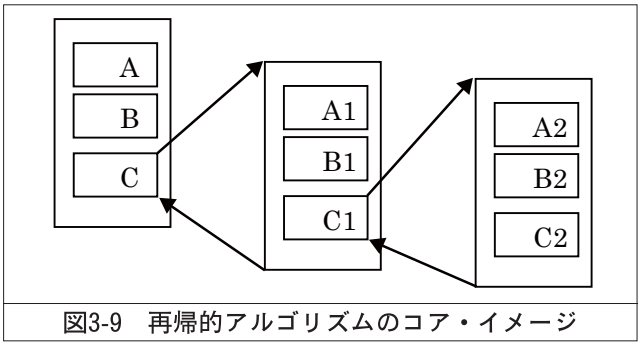

図3-9　再帰的アルゴリズムのコア・イメージ

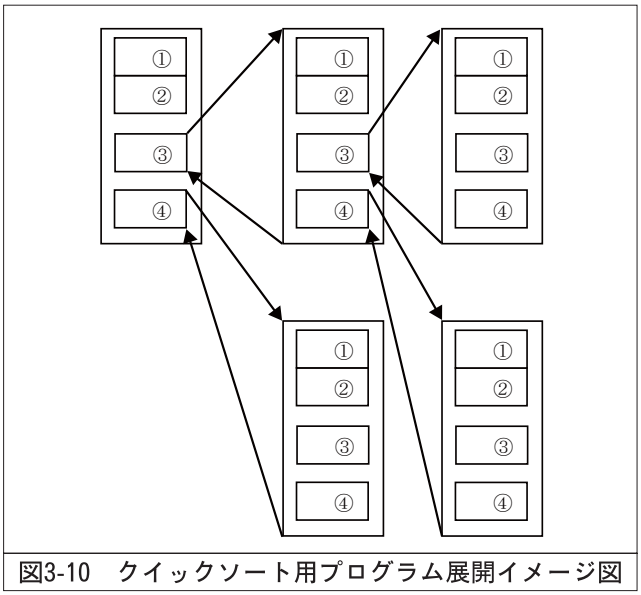

図3-10　クイックソート用プログラム展開イメージ図

3 - 2．数学（算数）的な問題のアルゴリズム

学生の多くは、「鶴亀算」や「濃度算」など「〜算」と聞いても、文章による問題の内容のみが想起されるだけで、アルゴリズム構造としての知識は持っていないように思われる。しかし、問題を解く段階になると、経験的に覚えているアルゴリズムを思い出しながら問題を解き始める。もちろん、このようなケースは、「〜算」という問題だけでなく、一般的な数学（算数）や理科の問題を解く場合でもあり得ることである。このようにアルゴリズム構造の知識なしで問題を解くということが起こる原因は、それまでにアルゴリズム構造の理解という意識がなかったからだと推測される。

そこで、「〜算」をアルゴリズム教育の授業の中で取り上げるとき、アルゴリズムを使って問題を解くだけではなく、そのコア・イメージによりアルゴリズム構造を理解し、それを利用して問題が解けるようになることを試みている。

3 - 2 - 1．鶴亀算のアルゴリズム

鶴亀算は、「単位量が異なる2つのものがあり、2つを合わせたものの全体個数と全体量（合計）が与えられ、それぞれのものの個数を求める」という問題である。例えば、「鶴と亀が合計20匹（羽）いる。足の数は全部で72本あった。鶴と亀は何匹いるか？」という問題である。「鶴亀算」という言葉は、この種の問題の代名詞的な意味で使われているだけである。一般的には方程式を使って解くが、アルゴリズム構造の理解という観点から、方程式を使わない方法を中心に扱っている。もちろん、方程式で解く方法についても説明はする。

⑴ 鶴亀算の例1

まず、代表的な鶴亀算で、上述した「鶴と亀が合計20匹（羽）いる。足の数は全部で72本あった。鶴と亀は何匹いるか？」という問題について考える。

（問題の背景とアルゴリズムの基本的な考え方）

問題の背景は、「2つの異なるものの単位量（この例では鶴と亀の足の数で暗黙の量となっている）、全体個数（頭の数）、全体量（足の数）の4つの値を使って、鶴と亀のそれぞれの個数を求める」ということである。アルゴリズムの基本的な考え方は、まず、単位量の差と、すべて鶴であると仮定した場合の量と実際の全体量との差に注目して、亀の個数を求めることである。ただし、すべて亀と仮定した考え方もある。

（コア・イメージの説明）

図3-11は、授業の中で提示していた鶴亀算のコア・イメージを改良したものである。本質的な部分は変わらないので、以下ではこのコア・イメージを使って説明する。

まず、与えられた4つの値と求める2つの値の関係が、構造的に理解できればよい。図の縦軸方向が単位量、横軸方向が個数を表し、図の中央の太線を中心に左が鶴、右が亀の個数を表す。単位量と個数の単位は、それぞれ「数／個：足の数」「個：頭の数」であり、「単位量×個数」が四角の個数になり、足の数に対応する。

この図は、鶴亀算のアルゴリズム構造の一般的な形で、簡単に言えば逆L字の構造であ

る。単位量が異なる２つのものが、それぞれ左右の位置に分かれて並列する形である。この構造は、次に述べる流水算も同じであるが、与えられる値（全体量や個数）が違うので、アルゴリズムは異なる。

（具体的なアルゴリズム）

上述した基本的な考え方に基づいた具体的なアルゴリズムは、次のようになる。以下では、単位量（単位あたりの足の数）の小さい方が鶴、大きい方が亀である。

①単位量差
　２つのものの単位量（単位あたりの足の数）の差を、単位量差として求める。
②仮の全体量（図の点線で囲った(A)部分の足の数）
　すべて小さい方（鶴）と仮定して、仮の全体量を求める。
③仮の差（図の点線で囲った(B)部分の足の数）
　仮の差＝与えられた全体量－仮の全体量
④大きい方（亀）　　大きい方＝仮の差 ÷ 単位量差
⑤小さい方（鶴）　　小さい方＝全体個数－大きい方

（アルゴリズムに基づく解）

上述したアルゴリズムに具体的な数値を代入して解を求めると、次のようになる。

①（単位量差）4－2＝2
②（仮の全体量）　2×20＝40
③（仮の差）72－40＝32
④（大きい方：亀）32÷2＝16
⑤（小さい方：鶴）20－16＝4

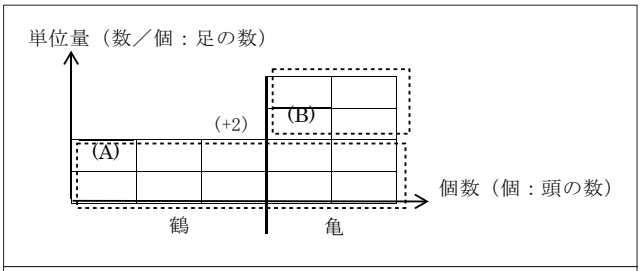

図3-11 鶴亀算のコア・イメージ

⑵　鶴亀算の例２

次に、「150円と180円のボールペンを合計30本買い、5100円払った。150円と180円のボールペンをそれぞれ何本買ったか？」という問題について考える。

図3-12は、図3-11のコア・イメージから作成した、この問題のイメージ図である。図3-11のコア・イメージとの違いは、与えられる単位量と全体量であり、ともに個数ではなく、量としてのお金である。この違いはアルゴリズムとしての違いにはならないので、この問題が鶴亀算であることが、この図から判断できる。

大きい方が180円、小さい方が150円、全体量が5100円であることに注意して、アルゴリズムにしたがって解けば、次のようになる。

①（単位量差）180－150＝30
②（仮の全体量） 150×30＝4500
③（仮の差）5100－4500＝600
④（大きい方：180円）600÷30＝20（本）
⑤（小さい方：150円）30－20＝10（本）

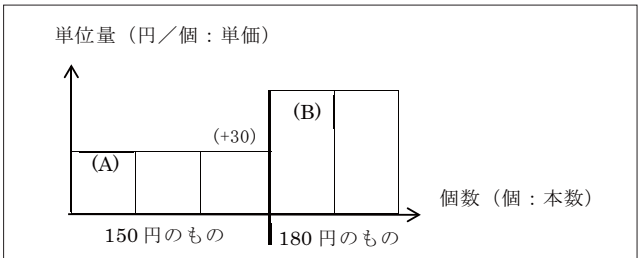

図3-12 鶴亀算のイメージ図 1

図3-11と図3-12を比較すると、アルゴリズム構造（問題の構造）が全く同じであることがわかる。さらに、この図3-12を図3-13のように書くと、鶴亀算の問題は、簡単な図形のアルゴリズム構造（問題の構造）であることがわかる。つまり、全体量を面積と考え、逆L字全体の面積と太線で示した線分の長さが与えられているとき、矢印の 2 つの長さを求めるという問題である。このような捉え方ができるのは、コア・イメージによってアルゴリズム構造が明確にできるからであると考えられる。

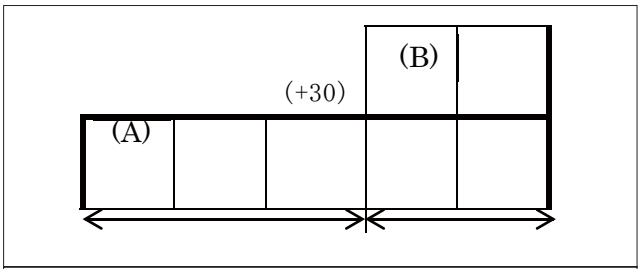

図3-13 鶴亀算のイメージ図2

⑶ 鶴亀算の例 3

次に、まえがきで示した問題「A地点からB地点に車で行くのに、一般道路を平均時速40km、高速道路を平均時速80kmで移動したところ、 4 時間で到着し、走行距離は220kmであった。一般道路と高速道路をそれぞれ何時間走ったか？」について考える。

図3-14は、例 2 と同様にコア・イメージから作成した、この問題のイメージ図である。図3-11のコア・イメージや図3-12のイメージ図と全く同じ構造であることがわかる。単位

量が速度、個数が時間になるだけで、典型的な鶴亀算であることが、この図から判断できる。

大きい方が高速道路（時速80km）、小さい方が一般道路（時速40km）、全体個数(時間)が４時間、全体量が走行距離220kmであることに注意して、アルゴリズムにしたがって解けば、次のようになる。

①（単位量差）80－40＝40
②（仮の全体量）　40×4＝160
③（仮の差）220－160＝60
④（大きい方：高速道路）60÷40＝1.5（時間）
⑤（小さい方：一般道路）4－1.5＝2.5（時間）

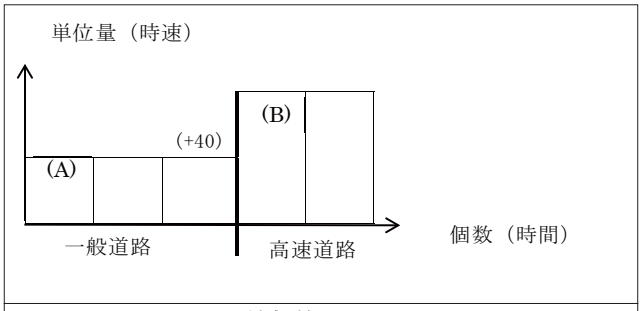

図3-14 鶴亀算のイメージ図3

３－２－２．流水算のアルゴリズム

流水算は、「静水時の船速と川の流速が与えられ、船で２点間を上下往復するときの所要時間を求める」というような問題である。例えば、「静水時の速さが時速８kmの船で、川下のAから川上のBまで30kmの距離がある２地点を往復するのに何時間かかるか？ただし、川の流速を時速２kmとする」というような問題である。流水算は、上りと下りの速度が違うことを考慮すれば、「速度×時間＝距離」をテーマとした問題と同じである。しかし、このアルゴリズムを取り上げる理由は、この流水算のコア・イメージが、鶴亀算と同じような構造になるからである。ここでは、鶴亀算との違いにも注目して考察する。

⑴　流水算の例１

上述した流水算の問題「静水時の速さが時速８kmの船で、川下のAから川上のBまで30kmの距離がある２地点を往復するのに何時間かかるか？ただし、川の流速を時速２kmとする」について考える。

（問題の背景とアルゴリズムの基本的な考え方）

問題の背景は、「２つの単位量（船速と流速)、移動する２点間の距離の３つが与えられ、上りと下りに要する所要時間を求める」ということである。アルゴリズムの基本的な考え方は、「速度×時間＝距離」の式に基づく。図形で考えると面積と縦の辺の長さが与えられたとき、２つの横の辺の長さを求めてその２つの合計を求めることである。

（コア・イメージの説明）

図3-15は、流水算のコア・イメージである。アルゴリズム教育の中で提示したものとは違うが、前述したように鶴亀算のコア・イメージと同じであることに注目したので、以下ではこれで説明する。

ここでも、鶴亀算と同様に、与えられた値の関係が構造的に理解できればよい。図の縦軸方向が上下の実質的な船速、横軸方向が時間を表し、図の中央の太線を中心に左が上り、右が下りの状態である。船速の単位は「距離／時間」となるので、「船速×時間＝距離」となり、図の左右の長方形の面積が距離を表す。したがって、船が往復する流水算では左右の長方形の面積は同じとなる。図の破線は静水時の船速を表し、破線の上下の等号（＝）のついた2つの辺の長さは流速を表す。この図に注目すると、次の4つの式が理解できる。

- 上り（の速さ）＝船速 － 流速
- 下り（の速さ）＝船速 ＋ 流速
- 下り＋上り＝ 船速 × 2
- 下り－上り＝ 流速 × 2

なお、第3式と第4式は、第1式と第2式からも導ける。流水算の問題では、この第3式と第4式を利用すると解答が容易になる場合がある。

ここで、鶴亀算の例3のイメージ図（図3-14）と流水算（図3-15）のコア・イメージを比較すると、両者の構造が同じであることがわかる。両者の違いは、与えられる全体量（距離）である。左右の全体量（2つの長方形の面積に相当）が与えられる流水算の方が、問題としては簡単であることがわかる。ここでは省略するが、両者の違いを明確にするには、それぞれのイメージ図に与えられた数値を明示すればよい。

（具体的なアルゴリズム）

次のような表を作成し、以下のアルゴリズムにしたがって、与えられる値や求まる値を表に埋める方法で解ける。また、コア・イメージにそれぞれの値を書き込んでいく方法でも解ける。

	船速→	流速↓	速度ｓ	時間ｔ	距離ｄ
下り↓	ｘ	ｙ	ｘ＋ｙ		
上り↑	ｘ	ｙ	ｘ－ｙ		

① 静水時の船速、流速、距離など問題で与えられているものを記入する。
②「上り＝船速 － 流速」「下り＝船速 ＋ 流速」により実質的な船の速度を求める。
③「速度×時間＝距離」を使って残りの表を埋める。
④ 時間の合計などを求める。

（アルゴリズムに基づく解）

上述したアルゴリズムにしたがって、具体的な数値を代入しながら、次の表を作成していく。なお、表中の番号（①②など）は、アルゴリズムの番号に対応する。

	船速	流速	速度	時間	距離
下り	①8	①2	②10	③3	①30
上り	①8	①2	② 6	③5	①30

（合計）　　④3 ＋ 5 ＝ 8（時間）

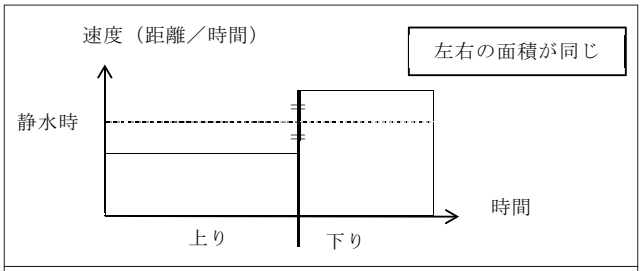

図3-15　流水算のコア・イメージ

(2) 流水算の例 2

次に、「静水時の速さが時速15kmの船がある。この船で、川上のA地点から川下のB地点まで行くのに 4 時間かかった。川の流れの速さが時速 5 kmだとすると、この船でB地点からA地点に行くのに何時間かかるか？」の問題について考える。

上述したアルゴリズムにしたがって、次のような手順で表を作成していけばよい。

	船速	流速	速度	時間	距離
下り	①15	①5	②20	①4	③80
上り	①15	①5	②10	④8	③80

例 1 の問題との違いは、距離ではなく所要時間が与えられていることである。しかし、「速度×時間＝距離」の式から、距離と時間のどちらか一方が与えられれば他方が求まるので、本質的には同じ問題である。

(3)　流水算の例 3

ここでは、上述した流水算の例 1 を少し変えた問題について考える。例えば、「静水時の速さが時速 8 kmの船で、川下のAから川上のBまで行き、さらにA-B間の途中にあるCまで移動した。走行距離は50kmで所要時間は 7 時間であった。A-B間とB-C間に要した時間はそれぞれ何時間か？ただし、川の流速を時速 2 kmとする」という問題である。

流水算のコア・イメージからこの問題のイメージ図を書くと図3-16のようになる。例 1 の問題と違うのは、上りと下りの走行距離が異なることと、与えられるものが全体の移動距離と所要時間であることである。与えられたものに注目して図3-16を見ると、この問題

は鶴亀算の例３と全く同じアルゴリズム構造であることが理解できる。したがって、鶴亀算のアルゴリズムが使えることがわかる。

大きい方を下り（時速10km)、小さい方を上り（時速６km)、全体個数を所要時間の７時間、全体量を走行距離50kmであることに注意して、アルゴリズムにしたって解けば、次のようになる。

①（単位量差）10－6＝4
②（仮の全体量） 6×7＝42
③（仮の差）50－42＝8
④（大きい方：下りB-C間）8÷4＝2（時間）
⑤（小さい方：上りA-B間）7－2＝5（時間）

この例のように問題を見ただけでは捉えにくいが、コア・イメージによりその構造に注目すると、どの構造と同じであるかがわかる。その結果、どのアルゴリズムが使えるかがわかる。このようなことが、アルゴリズムの応用につながると考えられる。

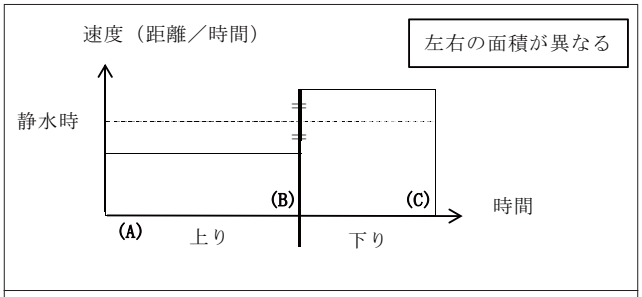

図3-16　流水算のイメージ図

３－２－３．濃度算のアルゴリズム

濃度算は、「５％の食塩水400gと９％の食塩水600gを混ぜると何％の食塩水になるか？」というような問題である。濃度算は、それぞれ異なる状態にある２つ以上のものを一つにまとめたときの状態をテーマとした問題である。大きく２つに分類できる。「２つ以上のものを一つにまとめたときの状態を求める問題（パターン１）」と「いくつかのものを一つにまとめたときの状態を与えて、初期状態のものを求めさせる問題（パターン２）」の２つのパターンである。上述した問題は、パターン１の典型的な問題である。

⑴　濃度算の例１（パターン１）

まず、上述したパターン１の問題「５％の食塩水400gと９％の食塩水600gを混ぜると何％の食塩水になるか？」について考える。

（問題の背景とアルゴリズムの基本的な考え方）

上述した２つのパターンのいずれの場合も、個々の状態を式で表し、まとめた状態を合計で求める。アルゴリズムの基本的な考え方は、２つ以上のそれぞれの状態を「食塩水（g)×濃度（％）＝食塩（g)」の式で表し、その合計を下段に求めて表形式にまとめるこ

とである。どちらのパターンも、「食塩水×濃度＝食塩」の式に基づく合計行から導かれる一次方程式を解くだけである。

（コア・イメージの説明）

図3-17は、濃度算のコア・イメージである。異なる状態にある2つ以上のものを縦に並べ、その横に対応する個々の状態「食塩水×濃度＝食塩」を書き、最後に食塩水と食塩の合計を求めるというアルゴリズム構造である。表構造と考えてよい。この図は2つの場合であるが、3つ以上の場合でも全く同じ構造となる。

（具体的なアルゴリズム）

コア・イメージを使ったアルゴリズムを具体的に示すと、次のようになる。

①「食塩水(g)×濃度(%)＝食塩(g)」に基づき、図3-18に示す表を作成する。そのとき、パターン2で求めるもの（未知数）があるとき、それをxとおいて式を立てる。

② 食塩水と食塩の合計を求める。

③ 最終的な合計行から導かれる一次方程式により、未知数の値を求める。

（アルゴリズムに基づく解）

図3-18は、コア・イメージを使ったアルゴリズムから求める解答例である。実際には、次の表のように図の右にある3行からなる表（見出しは除く）を作成し、合計行から導かれる一次方程式を解く。

食塩水	（×）濃度	（＝）食塩
400	5 / 100	20
600	9 / 100	54
1000	X	74

1000 × X ＝ 74

X ＝ 74／1000 ＝ 0.074 （7.4%）

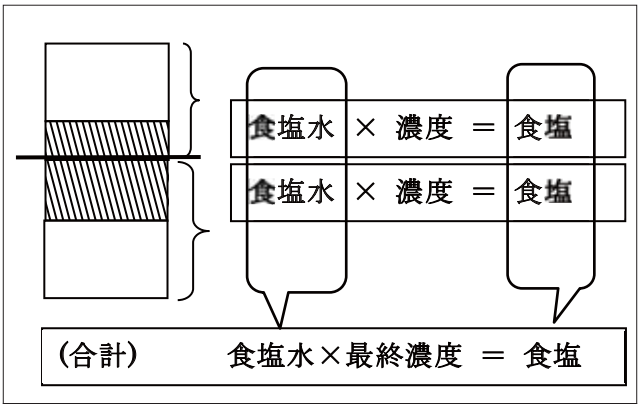

図3-17　濃度算のコア・イメージ

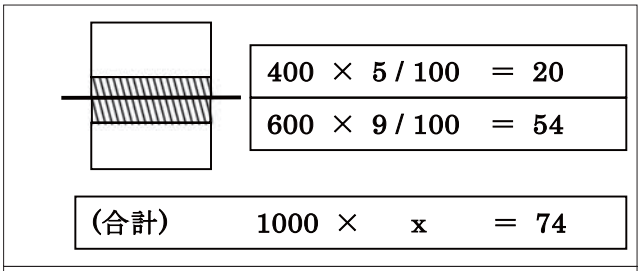

図3-18　濃度算の解答例

また、図3-19は、濃度算の特別な場合で、水だけ加える場合と食塩だけ加える場合のイメージ図である。水だけ加える場合は濃度0%、食塩だけを加える場合は濃度100%とすると、上述したアルゴリズムがそのまま使える。

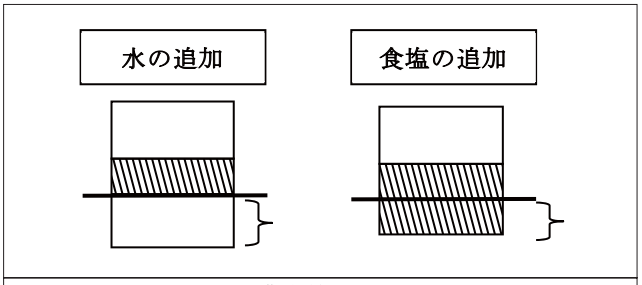

図3-19　濃度算のイメージ図

⑵　濃度算の例２（パターン２）

次に、パターン２の問題「２％の食塩水300gに９％の食塩水を混ぜて７％の食塩水を作るには、９％の食塩水を何ｇまぜればよいか？」について考える。図3-20は、コア・イメージを使ったアルゴリズムから求める解答例である。実際には、前と同じように、右の３行からなる次のような表を作成し、合計行の一次方程式を解く。

食塩水	（×）濃度	（＝）食塩
300	2 / 100	6
X	9 / 100	(9 / 100)X
300＋X	7 / 100	6 ＋ (9 / 100)X

(300＋X) × 7 / 100 ＝ 6 ＋ (9 / 100)X
7(300＋X) ＝ 600 ＋ 9X
2X ＝ 1500
X ＝ 750（ｇ）

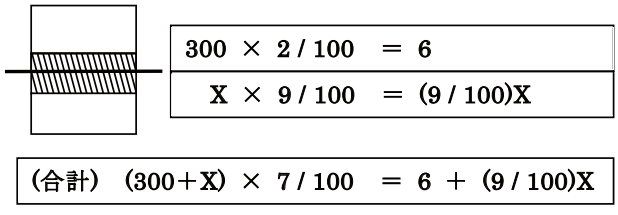

図3-20 濃度算の解答例

3 - 2 - 4．n進法のアルゴリズム

n進法の問題は、大きく「n進法→10進法（パターンn10)」と「10進法→n進法（パターン10n)」の2つに分類できる。例えば、パターンn10は、「2進法の101011は10進法ではいくつになるか？」、パターン10nは、「10進法で43は2進法ではいくつになるか？」という問題である。どちらのパターンの問題でも、一般的にはアルゴリズム化された計算式で求められる。しかし、計算式の意味と計算手順をアルゴリズムとして理解するためには、n進法のコア・イメージを使って、アルゴリズム構造を理解し、アルゴリズムの本質的な理解が不可欠と考えられる。ここでは、濃度算のコア・イメージとの関連も視野に入れて考える。

⑴　n進法の例1（パターンn10：n進法→10進法）

まず、n進法の数を10進法に変換する問題「2進法の101011は10進法ではいくつになるか？」について考える。

（問題の背景とアルゴリズムの基本的な考え方）

n進法を理解するには、n進法用の入れ物がどのような構造になっているかを理解すればよい。入れ物を理解するには、次に述べるコア・イメージを参照しながら、その考え方が理解できればよい（コア・イメージの説明参照)。

各桁の位を各列に対応させて考える。各列にはn－1段の棚しかないので、その列に載せられるべきパックがn個になったら、n個分をパックにして、その左の列の棚に移すことになる。いわゆる桁上がりである。(n^{k+1})は(n^k)をn個まとめてパックにしたものであるから式で表すと、

$$(n^{k+1}) = \{(n^k), (n^k), \cdots (n^k)\}\ (k = 0,1,2 \cdots)\quad (n^0=1)$$

となる。具体的には、$(123)_n$（n進法で123）は、(n^2)のパックが1個、(n)のパックが2個、(1)のパック が3個あることを表す。n進法では、各列の棚に何個のパックが載っているかを表す必要があるので、0～n－1のn個の数字（記号）が必要となる。

各桁の状態を濃度算のコア・イメージと同様に個々の状態と考えれば、その状態を表形式で表すことができるので、各桁の状態からその桁の個数を計算し、最後に合計を求めるというアルゴリズムで考えればよい。

（コア・イメージの説明）

図3-21は、ｎ進法のコア・イメージである。以下、この図を使って説明する。n進法の問題は、あるもの（例えば、コイン）がn進法用の入れ物にどのような状態で入っているのか（パターンn10)、あるいは、入るのか（パターン10n）を具体的に頭に思い浮かべることができればよい。n進法の入れ物としてのコア・イメージの意味を理解できれば、アルゴリズム構造も理解できると考えられる。

図3-21に示すn進法用の全体の入れ物は、ｎ－1段の棚からなる入れ物が、右から必要な列の数だけ並んだものである。図の各列はｎ進法の各桁に対応し、右端が１桁目である。各列は最大ｎ－1段の棚からなり、各列の段にはその列用のパックが載り、n進法の各桁の数字が対応する段に載っているパックの個数を表す。「(1)は１個のパック」「(n）は、(1)をn個まとめてパックにしたもの」、「(n^2）は、(n）をn個まとめてパックにしたもの」を表す。なお、パックについては、前述した基本的な考え方で説明したとおりである。

（具体的なアルゴリズム）

濃度算と同様に、各桁の状態を表形式にまとめればよいので、 次のような表で求める。具体的には、(a_k•••a_3 a_2 a_1 a_0)$_n$に対して、下位の桁(右の桁)から順に、その桁の数に１, n, n^2 … を掛けて合計を求める。

n^k	•••	n^3	n^2	n^1	n^0(=1)
a_k	•••	a_3	a_2	a_1	a_0
$n^k \times a_k$	•••	$n^3 \times a_3$	$n^2 \times a_2$	$n^1 \times a_1$	$1 \times a_0$

（アルゴリズムに基づく解）

「２進法の101011は10進法ではいくつになるか？」という問題の場合は、次のようになる。

2^5(32)	2^4(16)	2^3(8)	2^2(4)	2^1(2)	2^0(1)
1	0	1	0	1	1
32 (32×1)	0 (16×0)	8 (8×1)	0 (4×0)	2 (2×1)	1 (1×1)

101011＝32×1＋8×1＋2×1＋1＝43

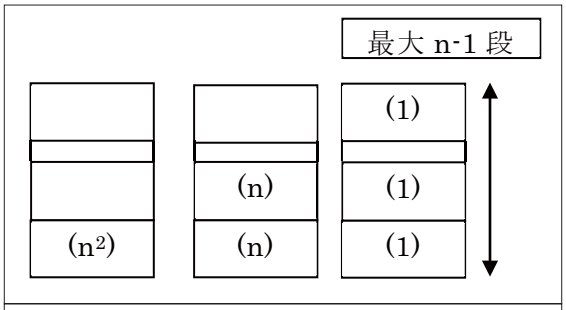

図3-21　n 進法のコア・イメージ

⑵　n 進法の例 2 （パターン10n：10進法→ n 進法）

次に10進法をn進法に変換する「10進法で43は 2 進法ではいくつになるか？」の問題について考える。この問題は、10進法で表わされる個数のものを、コア・イメージのn進法の入れ物に入れた場合、どのような状態になるかという問題である。

基本的な考え方は、10進法をnで割った場合、「その商はn個のパックにまとめられるパック数」「余りが各桁に載るパック数」を表すということである。具体的には、最初に与えられた10進法をnで割った場合、商が(n)の個数、余りが(1)の個数で下 1 桁目の数を表す。次に、商である(n)の個数をnで割ると、商が(n^2)の個数、余りが(n) の個数で下 2 桁目の数を表す。以下同様に、商がnより小さくなるまで、商をnで割る計算を繰り返せばよい。まとめると次のようになる。

① 10進法の数値をnで割り、商 1 と余り 1 を求める
　→　商 1 ＝(n)のパック数、余り 1 ＝(1)のパック数
　→　余り 1 が、n進法下 1 桁目の数
② 商 1 をnで割り、商 2 と余り 2 を求める
　→ 商 2 ＝(n^2)のパック数、余り 2 ＝(n)のパック数
　→ 余り 2 が、n進法下 2 桁目の数
③ 以下同様に、商をnで割り、
　「商 ＜ n」になるまで(商k)と(余りk)を求める
④ (最後の商)～ (余り 1)を並べたものが、求めるn進法の数となる

なお、最後の商もnで割ると、商0で最後の商がそのまま余りとなるので、それぞれの余りが、n進法用の入れ物の各列の棚に載るパックの個数と考えてもよい。

図3-22は、「10進法で43は 2 進法ではいくつになるか？」の問題を、上述したアルゴリズムに基づいて解いた計算過程である。したがって、解は「43 ＝ $(101011)_2$」となる。

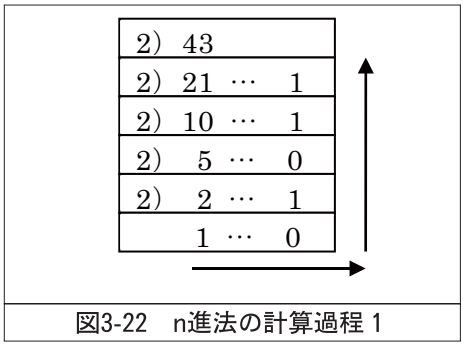

図3-22　n進法の計算過程 1

図3-23は、「10進法で472は 7 進法ではいくつになるか？」の問題を、上述したアルゴリズムに基づいて解いた計算過程である。したがって、解は「472 ＝ $(1243)_7$」となる。

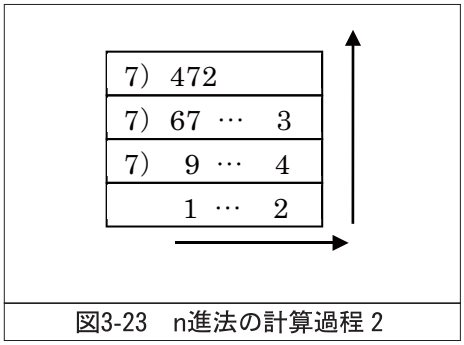

図3-23　n進法の計算過程 2

4．まとめ

従来のアルゴリズム教育では、アルゴリズム構造の理解は、それ程意識されていない。しかし、アルゴリズムの本質的な理解と応用のためには、それを使って問題を解くだけでなく、アルゴリズム構造の理解が不可欠と考えられる。筆者は、アルゴリズム構造の理解をそのコア・イメージの理解と位置づけ、コア・イメージ使った授業を試みている。この試みの成果については、厳密な評価・検証を行っていないので報告することはできないが、少なくとも学生の反応を見る限りにおいては、ある程度の成果はあるように思われる。厳密な評価・検証については、今後の課題としたい。

これまでの取り組みから、次のようなことが言える。

①アルゴリズム構造を直観的に理解できるコア・イメージの作成を工夫すると、その構造が明確になり、アルゴリズムの説明もしやすくなる。

②コア・イメージを描けることを、アルゴリズムの本質的な理解につなげられる場合がある。例えば、クイックソートやマージソートでは、コア・イメージを正確に書けることがアルゴリズムの本質的な理解になると考えられる。

③再帰的アルゴリズムを使ったプログラミング的アルゴリズムの場合、そのコア・イメージと再帰的アルゴリズムのコア・イメージをリンクさせると、捉えにくい再帰的アル

ゴリズムによる展開が具体的に把握できる。
④プログラミング的アルゴリズムの場合、プログラム構造またはデータの流れのどちらに焦点を置くかによって、異なるコア・イメージが考えられる。目的によって使い分ければよいが、併用することを考えてもよい。
⑤一見すると異なるように思える問題が、コア・イメージを元にそのイメージ図を作成することによって、同じアルゴリズム構造であることがわかる。鶴亀算の3つの例などである。
⑥逆に、一見すると同じように思える問題が、そのイメージ図を作成することによって、別のアルゴリズム構造であることがわかる。流水算の例3などである。
⑦鶴亀算や流水算が図形の問題として扱えることは、特に新しい考え方ではないが、このようなことがコア・イメージ作成の過程から必然的に意識されるようになることは、注目すべきことである。

コア・イメージを利用したアルゴリズム教育の成果の評価・検証をはじめ、今後の課題として、次のようなことが考えられる。
①コア・イメージを利用したアルゴリズム教育の成果の評価・検証を行う。
②検証の結果を踏まえ、コア・イメージのアルゴリズム教育への体系的な利用を考える。新しいアルゴリズム教育の提案ができる可能性がある。
③コア・イメージ作成のためのイメージ化の方法をさらに研究し、アルゴリズム構造の明確化についての方法論としてまとめる。
④アルゴリズム構造に着目し、アルゴリズムの類型化を図る。

参考文献

(1) 川口順功,「論理的データ構造とアルゴリズムのコア・イメージを利用したアルゴリズム教育」, 第26回ファジィシステムシンポジウム講演論文集, pp.989-992, 2010
(2) 川口順功, 渡邉志, 高橋等,「言葉の理解を意識した情報教育の試み」, 平成22年度情報教育研究集会論文集, pp.282-285, 2010
(3) 川口順功,「アルゴリズムのコア・イメージを使ったアルゴリズム教育」, 第27回ファジィシステムシンポジウム講演論文集, pp.1330-1333, 2011
(4) 川口順功, 高橋等, 永田奈央美, 大石義,「アルゴリズム的思考のビジュアル化によるラーニング構造の抽出」, 大学ICT推進協議会2011年度年次大会論文集, pp.282-287, 2011
(5) 西尾実, 岩淵悦太郎, 水谷静夫編,「岩波国語辞典第四版」, 岩波書店, 1986
(6) 田中茂範, 佐藤芳明, 河原清志,「イメージでわかる単語帳」, p.86, 日本放送出版協会, 2007
(7) 銀林浩,榊忠男,小沢健一編,「遠山啓エッセンス4 授業とシェーマと教具」, p.124, 日本評論社, 2009
(8) G.ポリア著, 柿内賢信訳,「いかにして問題をとくか」, 丸善出版, 1954
(9) 島内剛一, 有澤誠, 野下浩平, 浜田穂積, 伏見正則編,「アルゴリズム辞典」, 共立出版, 1994

(10) 柴田望洋，辻亮介，「C言語によるアルゴリズムとデータ構造」，ソフトバンクパブリッシング，2004
(11) 杉山行浩，「Cで学ぶデータ構造とアルゴリズム」，東京電機大学出版局，1995
(12) 近藤嘉雪，「Javaプログラマのためのアルゴリズムとデータ構造」，ソフトバンククリエイティブ，2011